Technische Universität Ilmenau
Fakultät für Mathematik
und Naturwissenschaften
Institut für Mathematik

Postfach 10 0565
98684 Ilmenau
Germany
Tel.: 03677/692652
Fax: 03677/691241
Telex: 33 84 23 tuil d.
email: W.Neundorf@mathematik.tu-ilmenau.de

Preprint No. M 11/96

Behandlung großer Matrizen auf dem PC

Werner Neundorf

Juni 1996

‡MSC (1991): 65-01, 65-04, 65F05, 65F50, 68-04, 68M07, 68Q25

1 Einleitung

In der Arbeit befassen wir uns damit, größere Matrizen mit bis zu mehreren Millionen Elementen unter Berücksichtigung ihrer Struktureigenschaften zu verarbeiten. Solche Matrizen sind zum Beispiel zu invertieren oder treten bei der Lösung von großen Gleichungssystemen und Eigenwertaufgaben auf.
Die konkrete Wahl einer Lösungsmethode wird entscheidend beeinflußt durch die hardwaremäßigen Gegebenheiten des verwendeten Rechners. Auf skalaren Rechnern wird dies zu anderen Ergebnissen führen als auf Hochleistungsrechnern, die Vektorinstruktionen verwenden oder sogar Parallelisierung ermöglichen.
Selbst auf skalaren Rechnern wird die Problemstellung von einer Vielzahl von Parametern beeinflußt, wie etwa :

- schnelle und große Rechenregister, Cache-Speicher,
- interne Parallelität des Rechenwerkes,
- Optimierer, die bestimmte Sprachkonstrukte sehr effektiv behandeln, andere aber nur weniger effektiv,
- Verwendung von Maschinencode,
- Fragen der ökonomischen Speicherung,
- Kapazität des Speichers mit schnellem Zugriff,
- Übertragungsgeschwindigkeit zwischen dem Zentralspeicher und den Hilfsspeichermedien, wie etwa Plattenspeicher.

Häufig wird die Rechenzeit des Programms/Algorithmus in direkten Zusammenhang mit der Anzahl der arithmetischen Operationen gebracht. Das ist ein wichtiges Kriterium. Aber im Zuge der Hardwareentwicklung sollte man andere Fakten keinesfalls unberücksichtigt lassen.
So ist zum z.B. eine Multiplikation beim i-Pentium genauso schnell wie eine Addition. Das Quadrieren einer Zahl ist gar doppelt so schnell wie die Addition. Zahlreiche verbesserte numerische Verfahren beruhen darauf, auf Kosten von Additionen einige Multiplikationen einzusparen (vgl. [18]). In [17] ist der Zeitgewinn untersucht, den diese Algorithmen erbringen. Dabei werden gewisse unrealistische Vorstellungen über den erzielbaren Gewinn häufig korrigiert. Einsparungen an Multiplikationen sind nicht mehr unbedingt sinnvoll, wenn eine Zuweisung mehr Zeit als eine Gleitkommamultiplikation benötigt (beim i-Pentium gar mehr als 200% der Zeit).
Dazu kommt die Unterstützung der Gleitkommaarithmetik (GKA) in der CPU durch spezielle Prozessoren. In der Sprache Pascal kann man durch Schalter (Compilerdirektive, Modus) oder Einstellungen im Menü (Menüäquivalent) festlegen, ob Operationen mit Realzahlen durch Routinen der Laufzeitbibliothek oder über die direkte Ansteuerung eines numerischen Koprozessors stattfinden.
Im Modus {$**N**-} steht nur der Datentyp *real* zur Verfügung, alle Operationen mit diesem Typ geschehen über Aufrufe der Laufzeitbibliothek. Genauer gesagt sind es aber keine Laufzeitroutinen, sondern spezielle Unterprogramme (call) im Kern von Pascal zur Multiplikation von zwei 6-Byte-Worten. Diese Art der Verarbeitung ist natürlich zeitaufwendiger. Der Typ *real* gehört auch nicht zum IEEE-Standard.

Im {$**N+**} Modus generiert Pascal Code für Gleitkommaberechnungen mit dem 80x87-Koprozessor und bietet weitere vier Real-Typen: *single, double, extended* und *comp*. Bei Einstellung der Option nutzt Pascal für diese den im Rechner vorhandenen 80x87-Koprozessor. Der Ablauf ist z.B. bei einer Multiplikation in *double* unter Anwendung von Koprozessorroutinen wie folgt :

- ***fld*** : Laden eines 8-Byte-Wortes (*double*-Zahl) in den Koprozessor,
- ***fmul*** : Multiplikation einer Zahl im Koprozessor mit einer Zahl aus dem Speicher,
- ***fst*** : Rücktransfer des Ergebnisses in den Speicher.

Ansonsten können auch die 80x87-Befehle von der Laufzeitbibliothek emuliert werden.
Nimmt man den Typ *real* im {$N+} Modus, dann laufen die Berechnungen über *double*, wobei durch Konvertierungen Zeitverluste auftreten. Die Reihenfolge der Ausführung einer Multiplikation ist im groben:

- Konvertierung von zwei 6-Byte-Worten (*real*-Zahlen) nach *double* und Speicherung im Koprozessor,
- Multiplikation dieser im Koprozessor mittels Routine ***fmul***,
- Rückkonvertierung in den Speicher im 6-Byte-Format.

In den Tabellen 5 und 6 sind die Zeitverluste bei der Behandlung von GKZ des Typs *real* mit/ohne Numerik-Koprozessor deutlich zu erkennen. Für viele akademische Beispiele nutzt man traditionell noch den *real*-Typ. Dabei ist es nicht ausschlaggebend, eine gezielte Auswahl der GKA zu treffen. Des weiteren wird die Standardausgabe von *real*-Zahlen im 17-stelligen Format "-x.xxxxxxxxxE+xx" insbesondere wegen dem Exponentenanteil zumeist angenehmer empfunden als das 23-stellige Format "-x.xxxxxxxxxxxxxxE+xxxx" (15-stellige Mantissenlänge passend für *double*) der anderen reellen Typen.

Aber für die Lösung von Problemen der Numerischen Mathematik ist die Einhaltung eines gewissen Standards der GKA, auch in Hinblick auf die Kombination verschiedener Programmiersprachen, sowie die schnelle Ausführung einer großen Anzahl arithmetischen Operationen natürlich wichtig.

Wir beschäftigen uns hier mit dem Problem, wie man für klassische Algorithmen mit großen Matrizen die Eigenschaften des Rechnersystems und/oder des Compilers nutzen kann. Wir untersuchen unter Einsatz von PC-Technik

- **die Invertierung einer Matrix**,
- **die Matrix-Vektor-Multiplikation mit sparsen Matrizen**.

Dabei werden wir vorhandene Verfahren an rechnerspezifische Gegebenheiten anpassen und verschiedene Varianten testen.

2 Besonderheiten des Rechnersystems und Programmiersprache

Sämtliche Tests wurden auf PC unter dem Betriebssystem MS-DOS gemacht. Es wurden folgende Rechner angewendet.

Konfiguration	ASI	i-Pentium
CPU Type	80486DX2-S	80486-S
Co-Processor	Installed	Installed
CPU Clock	66MHz	96MHz
Base Memory	640K	639K
Extended Memory	7168K	31744K
Cach Memory	256K	256K

Tab.1. Systemkonfiguration der PC

Als Programmiersprache diente **Borland Pascal** Version 7.0.

Im Hauptspeicher des PC mit dem 64KByte-Datensegment können nur kleindimensionierte Probleme gelöst werden. Bei Bearbeitung einer bzw. von zwei quadratischen Matrizen der Größe n x n und weiteren, aber wenigen Vereinbarungen kommt man für die verschiedenen Gleitkommaformate auf folgende Obergrenzen für die Dimensionen n.

Reeller Typ	Zahlenbereich abs(r)	Anzahl der Byte	Gueltige Dezimal-ziffern	Bit-anzahl der Mant.	1+x<>1 1+x/2=1 x	Matrizen 1 n	Matrizen 2 n
single	1.5E-45..3.4E38	4	7- 8	24	1.2E-7	127	90
real	2.9E-39..1.7E38	6	11-12	40	1.8E-12	104	73
double	5.0E-324..1.7E308	8	15-16	53	2.2E-16	90	63
extended	1.9E-4951..1.1E4932	10	19-20	64	1.1E-19	80	57
comp	0..9223372036854775807 =2^63-1	8	19-20	64			

Tab.2. Gleitkommaformate und Matrixgroessen im 64KByte-Datensegment

Man beachte auch den sogenannten **d-Bereich** bei den GK-Formaten *single*, *real*, *double* oder *extended*, der von der kleinsten positiven normierten reellen Zahl (1.Ziffer der Mantisse $\neq 0$) bis zur kleinsten positiven intern darstellbaren Zahl reicht. Es kommt also die Mantissenlänge im Exponenten hinzu.

Mit dem Heap-Speicher oder Filearbeit und externen Speicher ist eine beachtliche Vergrößerung der Dimension schon möglich. Bei Borland Pascal kann man den Zugriff auf den Heap durch einen Modus in der **Zielplattform** des Compile-Menü entsprechend einrichten. Somit hat man einen erheblichen Teil des RAM-Speichers zur Verfügung.

GKF	Real Mode	Protected Mode	
		ASI 8MB RAM	i-Pentium 32 MB RAM
	480KByte	5300KByte	29000KByte
single	120 000	1 320 000	7 200 000
real	80 000	880 000	4 800 000
double	60 000	660 000	3 600 000
extended	48 000	530 000	2 900 000

Tab.3. Verfügbarer Heap-Speicher der PC und Anzahl der speicherbaren Gleitkommazahlen (GKZ) in Borland Pascal (Angaben gerundet)

Wird Borland Pascal unter Windows benutzt, so wird der nutzbare Speicherplatz meistens um einige Megabyte reduziert.

Inzwischen gibt es Compiler, die es gestatten, das 64KByte-Datensegment zu sprengen. Die bisherige Adreßbildung mittels *Segment:Offset* bzw. *adresse:=seg*16+ofs* für einen Speicherbereich von 1MB wird dabei verändert auf die volle Nutzung der 4 Byte als Adresse im Zusammenhang mit dem 32Bit-Datenbus. Der adressierbare Speicher ist somit ein linearer Block der Länge von ca. 4GB. Ein solcher Compiler ist **Watcom** für DOS und Unix mit den Sprachen C bzw. C++ . Er läuft im Protected Mode mit dem Zusatz DOS/4GW des Betriebssystems DOS4. Watcom findet z.B. auch Anwendung im Programmsystem MATLAB.

2.1 Speicherung von Matrizen mittels Heap

Im Protected Mode und bei Nutzung des Heap-Speichers bleiben die Grenzen für Datenstrukturen durch das 64KByte-Datensegment weiterhin bestehen.
Die Größe reicht für einen Adreßvektor der Länge 16383, da

1 Adresse = 2 Word (Seg,Ofs) = 4 Byte.

Andererseits kann damit ein Vektor (z.B. die Zeile einer Matrix) definiert werden mit der Länge:

16383	Komponenten von Typ	*single*
10922	. . . . .	*real*
8191	. . . . .	*double*
6553	. . . . .	*extended.*

Für eine Rechteckmatrix T[1..16383,1..65535] mit Komponenten vom Typ *byte* würde man ca. 1100MB=1.1GB Heap brauchen. Bei Beibehaltung des Zeilenvektors, aber kleinerem Heap können natürlich nur entsprechend weniger Zeilen adressiert werden.

Wollte man die oben genannten Grenzen der Datenstrukturen maximal nutzen für die Speicherung einer (n,n)-Matrix von GKZ, so würde man folgenden Heapspeicher benötigen:

1100MB	bei *single*
720MB	bei *real*
540MB	bei *double*
430MB	bei *extended*.

Da solche Größenordnungen des RAM-Speichers im allgemeinen (noch) nicht zur Verfügung stehen, wird die Datensegmentgröße für den praktisch viel kleineren Heap also keine Beschränkung darstellen.
Den verfügbaren Heap-Speicher des PC kann man nun auch auf die maximale Dimension einer quadratischen Matrix $A(n,n)$ mit GKZ als Elemente umrechnen.

GKF	Real Mode	Protected Mode	
		ASI 8MB RAM	i-Pentium 32 MB RAM
	480KByte	5300KByte	29000KByte
single	346	1150	**2700**
real	280	940	2200
double	240	**810**	1900
extended	220	720	1700

Tab.4. Verfügbarer Heap-Speicher der PC und Dimensionen n der maximal speicherbaren Matrix von GKZ in Borland Pascal (Angaben gerundet)

Aus diesen Betrachtungen ist die nachfolgende Struktur der Vereinbarungen im Pascal-Programm abgeleitet worden. Ein Matrixzeiger zeigt auf einen Vektor von Zeigern (im Heap), mit denen ihrerseits die Zeilen der Matrix adressiert werden. Der zusätzliche Zeilenzeigervektor im Heap ist nicht speicherproblematisch, da er hier die Größe von $2700 * 4 \approx 10$KB nicht überschreitet (gemäß Format *single*, 32 MB RAM).

```
const nmax    = 810;
type  float   = double;
      vektor  = array[1..nmax] of float;
      Pvektor = ^vektor;
      matrix  = array[1..nmax] of Pvektor;
      Pmatrix = ^matrix;
var   n : integer;
      A : Pmatrix;
```

Bei der Implementierung der Matrixinvertierung im Kapitel 3 sind solche Datentypen und Variablen eingesetzt worden.

3 Matrixinvertierung

Die Bestimmung der Inversen der rellen Matrix $A(n,n)$ erfolgt mittels Austauschverfahren, auch Gauß-Jordan-Verfahren genannt, unter Nutzung des Heap. Die Transformation von A auf A^{-1} erfolgt dabei in n Schritten auf dem Platz mit einer Pivotstrategie (Spaltenpivotisierung mit Zeilenvertauschung), indem im Gleichungssystem $Ax = y$ spaltenweise sukzessiv die Variablen $x_1, x_2, ..., x_n$ eliminiert werden.

$$\begin{aligned} A &= A^{(0)} \to A^{(1)} \to ... \to A^{(n)} \\ Ax &= y \to (A^{(n)}P)\,y = x, \quad A^{-1} = A^{(n)}P, \quad P \text{ Permutationsmatrix} \end{aligned}$$

Beispiel: ohne Pivotsuche

$$A = A^{(0)} = \begin{pmatrix} \boxed{1} & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{pmatrix} \quad \begin{aligned} x_1+ \quad x_2+ \quad x_3 &= y_1 \to \quad x_1 = y_1 - x_2 - x_3 \\ x_1+ \quad 2x_2+ \quad 3x_3 &= y_2 \\ x_1+ \quad 3x_2+ \quad 6x_3 &= y_3 \end{aligned}$$

$$A^{(1)} = \begin{pmatrix} 1 & -1 & -1 \\ 1 & \boxed{1} & 2 \\ 1 & 2 & 5 \end{pmatrix} \quad \begin{aligned} y_1- \quad x_2- \quad x_3 &= x_1 \\ y_1+ \quad x_2+ \quad 2x_3 &= y_2 \to \quad x_2 = -y_1 + y_2 - 2x_3 \\ y_1+ \quad 2x_2+ \quad 5x_3 &= y_3 \end{aligned}$$

$$A^{(2)} = \begin{pmatrix} 2 & -1 & 1 \\ -1 & 1 & -2 \\ -1 & 2 & \boxed{1} \end{pmatrix} \quad \begin{aligned} 2y_1- \quad y_2+ \quad x_3 &= x_1 \\ -y_1+ \quad y_2- \quad 2x_3 &= x_2 \\ -y_1+ \quad 2y_2+ \quad 5x_3 &= y_3 \to \quad x_3 = y_1 - 2y_2 + y_3 \end{aligned}$$

$$A^{-1} = A^{(3)} = \begin{pmatrix} 3 & -3 & 1 \\ -3 & 5 & -2 \\ 1 & -2 & 1 \end{pmatrix} \quad \begin{aligned} 3y_1- \quad 3y_2+ \quad y_3 &= x_1 \\ -3y_1+ \quad 5y_2- \quad 2y_3 &= x_2 \\ y_1- \quad 2y_2+ \quad y_3 &= x_3 \end{aligned}$$

Bei Anwendung der Pivotstrategie entsteht der Permutationsvektor der Zeilenvertauschung $p = (1,3,2)$ (Pivotelement in der 2.Spalte von $A^{(1)}$ ist das Element 2 in der 3.Zeile). Daraus leitet sich die Matrix P ab. Es gilt dann

$$A^{-1} = A^{(3)}P = \begin{pmatrix} 3 & 1 & -3 \\ -3 & -2 & 5 \\ 1 & 1 & -2 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 3 & -3 & 1 \\ -3 & 5 & -2 \\ 1 & -2 & 1 \end{pmatrix}$$

Algorithmus des Austauschverfahrens mit Pivotstrategie am Platz

(1) Initialisierung des Permutationsvektors : $p_i = i,\ i = 1(1)n$

(2) Transformation in n Schritten : $k = 1(1)n$

(a) Pivotsuche in Spalte k : $|a_{qk}| = \max\limits_{k \le i \le n} |a_{ik}|$

Wenn $q > k$, vertausche die q-te und k-te Zeile von A sowie p_q und p_k.

(b) Falls $a_{kk} = 0$, dann ist Matrix singulär und STOP.

(c) Elimination der Variablen x_k und Umrechnung der Matrixelemente

$$\begin{aligned} h &= \frac{1}{a_{kk}} \\ a_{ik} &= h\, a_{ik}, \quad i = 1(1)n \\ a_{kk} &= h \\ j &= 1(1)k-1,\ k+1(1)n \\ a_{ij} &= a_{ij} - a_{ik}\, a_{kj}, \quad i = 1(1)k-1,\ k+1(1)n \\ a_{kj} &= -h\, a_{kj} \end{aligned}$$

(3) Spaltentausch auf der Basis von Permutationsvektor p mit Hilfsvektor v.

$$\begin{aligned} i = {} & 1, 2, ..., n \\ & v_{p_k} = a_{ik}, \quad k = 1(1)n \\ & a_{ik} = v_k, \quad k = 1(1)n \end{aligned}$$

Eine andere Variante der Spaltenvertauschung im Schritt (3) des Algorithmus benutzt anstelle von v den ganzzahligen Hilfsvektor ph für die Permutationen.

```
ph:=p;
repeat
 tausch:=true;
 for j:=n-1 downto 1 do
  if ph[j]<>j then
   begin
    tausch:=false;
    phj:=ph[j];
    for i:=1 to n do
     begin s:=a[i,j]; a[i,j]:=a[i,phj]; a[i,phj]:=s end;
    ph[j]:=ph[phj]; ph[phj]:=phj
   end
until tausch;
```

3.1 Matrixtest am PC

Der Aufwand (Zeitkomplexität) der Matrixinversion beträgt $T(n) = \mathcal{O}(n^3)$.
In mehreren Tabellen ist die Rechenzeit für die Inversion voll besetzter Matrizen mittels Austauschverfahren (vgl. [19]) zusammengestellt. Obwohl für numerische Berechnungen dieser Art der Gleitkommatyp *single* der Matrixelemente wenig relevant ist, wurde er an einigen Stellen jedoch aufgeführt. Die Rechenzeiten widerspiegeln die Formel der Zeitkomplexität.
Mit dem Zuwachsfaktor $q = \left(\frac{n_1}{n_2}\right)^3$ kann man auf Rechenzeiten für Inversionen von Matrizen weiterer Dimensionen schließen.

n	single	double	extended	real	
				\$N+	\$N-
50	0.49	0.49	0.66	1.16	1.65
100	4.01	4.62	5.55	9.72	13.18
200	31.70	35.10	48.06	78.92	106.23
300	114.69	126.82	168.35	286.38	383.70
400	273.42	296.71	406.45	665.59	895.28
720			$\approx 40\,min$		
810		$\approx 41\,min$			
940				$\approx 144\,min$	$\approx 194\,min$
1150	$\approx 108\,min$				

Tab.5. Rechenzeiten in *sec* für Matrixinversion auf PC ASI, 66MHz, BP Protected Mode, $nmax = n$ (Zeiten schwanken um 2%)

n	single	double	extended	real	
				\$N+	\$N-
50	0.11	0.17	0.22	0.38	0.55
100	1.70	1.76	2.53	3.68	4.77
200	15.00	15.49	22.52	31.68	40.75
300	52.01	53.99	78.21	108.37	140.00
400	101.29	117.81	189.93	254.09	318.46
720			$\approx 18\,min$		
810		$\approx 18\,min$			
940				$\approx 56\,min$	$\approx 72\,min$
1150	$\approx 49\,min$				

Tab.6. Rechenzeiten in *sec* für Matrixinversion auf PC i-Pentium, 96MHz, BP Protected Mode, $nmax = n$ (Zeiten schwanken um 2%)

	$nmax =$				
n	n	$2n$	$3n$	$4n$	$5n$
50	0.49	0.55	0.49	0.49	0.49
100	4.62	4.12	4.06	4.39	4.06
200	35.10	36.69	37.46	36.53	36.86

Tab.7. Rechenzeiten in *sec* für Matrixinversion auf PC ASI, 66MHz, BP Protected Mode, GKF *double*, $nmax = n, 2n, 3n, 4n, 5n$

Die Tabelle 7 zeigt, daß die vorsorgliche Reservierung von Speicherplatz auf dem Heap sich nicht auf die Rechenzeit auswirkt.

4 Matrix-Vektor-Multiplikation für sparse Matrizen

Die Behandlung von schwach besetzten Matrizen im Zusammenhang mit der
- Matrix-Vektor-Multiplikation,
- Lösung von Gleichungssystemen,
- Parallelisierung von Algorithmen mit Matrizen u.ä.

hat sich inzwischen zu einem fast eigenständigen Untersuchungsgebiet etabliert. Wichtige Zielstellungen sind dabei immer wieder die günstige Speicherung der von Null verschiedenen Matrixelemente (Nichtnullelemente, NNE) sowie die effiziente Implementierung der notwendigen Operationen.
So findet man in [3] einen guten Einstieg in die Problematik
- Besetztheitsstruktur einer Matrix,
- Bandbreitenreduzierung,
- Datenstrukturen und Kompaktspeichertechnik.

Hier wollen wir eine Variante der Kompaktspeichertechnik und ihre Umsetzung mittels entsprechender Datenstrukturen auf dem PC betrachten.
Bezüglich der Bereitstellung von sparsen symmetrischen Testmatrizen wird die Möglichkeit der Belegung der NNE auf der Basis der Elementstruktur eines FEM-Netzes einbezogen.

4.1 Kompaktspeichertechniken

Die kompakte Ablage der NNE einer Matrix erfordert weitere Informationen, aus denen man eindeutig die Position des Elements in der Matrix ablesen kann. Dies sind zusätzliche Indexvektoren. Für jedes NNE sind in der Regel drei Informationen zu speichern, wobei sein (reeller) Wert ($\neq 0$) hier eine untergeordnete Rolle spielt. Nimmt man für alle NNE den Wert 1 an, so kann man die Matrix z.B. als Adjazenzmatrix eines Graphen interpretieren.

Zunächst geben wir einige Kompaktspeichertechniken an. Ihre Veranschaulichung erfolgt jeweils an einer der folgenden Matrizen mit eingetragenen NNE (Leerstelle = Nullelement).

$$a_1 = \begin{pmatrix} a_{11} & & a_{13} & & & a_{16} \\ & a_{22} & & a_{24} & a_{25} & \\ a_{31} & & a_{33} & & & \\ & a_{42} & & a_{44} & & a_{46} \\ & a_{52} & & & a_{55} & \\ a_{61} & & & a_{64} & & a_{66} \end{pmatrix} = a_1^T, \quad a_2 = \begin{pmatrix} a_{11} & & a_{13} & & \\ & a_{22} & & & a_{25} \\ a_{31} & & & a_{34} & \\ & & a_{43} & a_{44} & \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix} \neq a_2^T$$

Im weiteren sei $a = a(n,n)$ eine quadratische Matrix mit reellen Elementen a_{ij}, n ihre Dimension und $nne \leq n^2$ die Anzahl aller ihrer NNE.
Der NNE-Vektor sei A, wobei $0 \neq a_{ij} \rightarrow A(l)$, die Indexvektoren sind mit J und I bezeichnet.

Ausgewählte Kompaktspeichertechniken:

(1) Zeilenweise Speicherung der Matrix
Demonstrationsmatrix : $a_1(n,n),\ n=6$

$A(1..nne),\ nne=16$, Vektor der NNE der Matrix (mit neuer Indizierung)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a_{11}	a_{13}	a_{16}	a_{22}	a_{24}	a_{25}	a_{31}	a_{33}	a_{42}	a_{44}	a_{46}	a_{52}	a_{55}	a_{61}	a_{64}	a_{66}

$J(1..nne)$ Vektor der zunehmenden Kolonnen- bzw. Spaltenindizes
$1 \le J(l) \le n$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	**3**	**6**	**2**	**4**	**5**	**1**	**3**	**2**	**4**	**6**	**2**	**5**	**1**	**4**	**6**

$I(1..n+1)$ Zeilenzeigervektor
$1 = I(1) \le I(2) \le ... \le I(n+1) = nne+1 \le n^2+1$
$I(i+1)-I(i)$ Anzahl der NNE in Zeile i
$g(i) = I(i+1)-I(i)-1$ Grad der i-ten Knotenvariable in FEM-Netz ($a_{ii} \neq 0$)

1	2	3	4	5	6	7
1	**4**	**7**	**9**	**12**	**14**	**17**

(2) Zeilenweise Speicherung der unteren Hälfte einer symmetrischen Matrix
Demonstrationsmatrix : $a_1(n,n),\ n=6,\ a_{ii} \neq 0,\ a_{ij}=a_{ji}$

$A(1..nneu),\ nneu=(nne+n)/2=11$

a_{11}	a_{22}	a_{31}	a_{33}	a_{42}	a_{44}	a_{52}	a_{55}	a_{61}	a_{64}	a_{66}

$J(1..nneu)$ Vektor der zunehmenden Spaltenindizes, $1 \le J(l) \le n$

1	2	1	3	2	4	2	5	1	4	6

$I(1..n)$ Zeilenzeigervektor
$1 = I(1) < I(2) < ... < I(n) = nneu,\ I(0) := 0$
$I(i)-I(i-1)-1 + \{Anzahl\ der\ J(l)\ mit\ J(l)=i\}$ Anzahl der NNE in Zeile i

1	2	4	6	8	11

(3) Zeilenweise Speicherung der Matrix mit Zeilen- und Spaltenindizes
Demonstrationsmatrix : $a_2(n,n),\ n=5$

$A(1..nne),\ nne=11$

a_{11}	a_{13}	a_{22}	a_{25}	a_{31}	a_{34}	a_{43}	a_{44}	a_{53}	a_{54}	a_{55}

$J(1..nne)$ Vektor der Spaltenindizes der NNE, $1 \leq J(l) \leq n$

1	3	2	5	1	4	3	4	3	4	5

$I(1..nne)$ Vektor der Zeilenindizes der NNE, $1 \leq I(l) \leq n$

1	1	2	2	3	3	4	4	5	5	5

(4) Spaltenweise Speicherung der Matrix mit Zeilenindizes und Spaltenbeginn-/Spaltenzeigervektor
Demonstrationsmatrix : $a_2(n,n),\ n = 5$

$A(1..nne),\ nne = 11$

a_{11}	a_{31}	a_{22}	a_{13}	a_{43}	a_{53}	a_{34}	a_{44}	a_{54}	a_{25}	a_{55}

$I(1..nne)$ Vektor der Zeilenindizes der NNE, $1 \leq I(l) \leq n$

1	3	2	1	4	5	3	4	5	2	5

$J(1..n)$ Vektor der Indizes derjenigen Komponenten von I, mit denen eine neue Spalte beginnt, $1 \leq J(l) \leq nne,\ J(n+1) := nne + 1$

1	3	4	7	10	12

(5) Zeilenweise Speicherung der Matrix mit Spaltenindizes und Zeilenbeginn-/Zeilenzeigervektor
Demonstrationsmatrix : $a_2(n,n),\ n = 5$

$A(1..nne),\ nne = 11$

a_{11}	a_{13}	a_{22}	a_{25}	a_{31}	a_{34}	a_{43}	a_{44}	a_{53}	a_{54}	a_{55}

$J(1..nne)$ Vektor der Spaltenindizes der NNE, $1 \leq J(l) \leq n$

1	3	2	5	1	4	3	4	3	4	5

$I(1..n)$ Vektor der Indizes derjenigen Komponenten von J, mit denen eine neue Zeile beginnt, $1 \leq I(l) \leq nne,\ I(n+1) := nne + 1$

1	3	5	7	9	12

Das entspricht also der Variante (1).

(6) Spaltenweise Speicherung der Matrix mit Speicherabbildungsfunktion
dazu Position eines Elements a_{ij} innerhalb des Vektors A
Demonstrationsmatrix : $a_2(n,n),\ n=5$

$A(1..nne),\ nne=11$, NNE spaltenweise

a_{11}	a_{31}	a_{22}	a_{13}	a_{43}	a_{53}	a_{34}	a_{44}	a_{54}	a_{25}	a_{55}

$S(1..nne)$ Vektor der Werte der Speicherabbildungsfunktion für NNE
$S(l)=i+(j-1)\,n$ Nummer von a_{ij}

1	3	7	11	14	15	18	19	20	22	25

Für diese Kompaktspeicherstrategien sind Such- und Einsortieralgorithmen ziemlich aufwendig, da entweder das ganze Feld durchgemustert werden muß bzw. viele Komponenten um eine Position nach hinten verschoben werden müssen zwecks Freimachung einer Stelle.
Will man diese beiden Aspekte etwas geschickter handhaben, so bietet sich die Speichertechnik mit verketteten Listen an. Die verschiedenen Versionen ergeben sich dann aus einfach oder doppelt verketteten Listen bzw. spalten- oder zeilenweiser Verkettung (siehe [3]).

Hier wenden wir die Variante (1) an und notieren noch einmal in zusammengefaßter Form die von der Matrix a abgeleiteten Vektoren.

$a(n,n)$	quadratische Matrix der Dimension n
nne	Anzahl der NNE der Matrix, $nne \le n^2$
$A(1..nne)$	Vektor der NNE zeilenweise, $a_{ij} \to A(l)$
$J(1..nne)$	Vektor der zunehmenden Spaltenindizes, $1 \le J(l) \le n$
$I(1..n+1)$	Zeilenzeiger-/Zeilenbeginnvektor Vektor der Indizes derjenigen Komponenten von J, mit denen eine neue Zeile beginnt $1 = I(1) \le I(2) \le ... \le I(n) \le I(n+1) = nne+1$ $I(i+1)-I(i)$ Anzahl der NNE in Zeile i

4.2 Generierung einer sparsen Matrix

Neben der Vorgabe akademischer Beispiele für die Matrix-Vektor-Multiplikation mit sparsen Matrizen wurde programmseitig ein Modul eingebaut, das eine Vernetzungsstruktur eines Gebietes, wie sie in der Methode der finiten Elemente auftritt, in die NNE-Struktur und weiter auf die Vektoren A, J, I transformiert. In der Menüführung des Programms ist diese Variante ersichtlich. Sie erfordert natürlich Dateiarbeit.
Das FEM-Netz liegt entweder in Form einer Textdatei vor oder ist als solche einzugeben. Die NNE haben den Wert 1.
Die Struktur des Netzfiles ist folgende (vergl. [4]).

n	Dimension der Matrix, Anzahl der Knotenpunkte/Freiheitsgrade
$nknot$	Anzahl der Knotenpunkte pro Element der 1.Gruppe
np_i	$i = 1(1)\,nknot$ Knotennummern im Element
np_i	$-\,\|\|\,-$
...	
-1	$(np_1 = -1)$ Schlußzeile für Elementtyp der 1.Gruppe
$nknot$	Anzahl der Knotenpunkte pro Element der 2.Gruppe
np_i	$i = 1(1)\,nknot$ Knotennummern im Element
...	
-1	$(np_1 = -1)$ Schlußzeile für Elementtyp der 2.Gruppe
...	
...	
0	$(nknot = 0)$ Schlußzeile der Textdatei

Die Transformation auf das NNE-File erfolgt über einen Zwischenschritt.
Dabei werden die Indexpaare (i, j) der NNE oberhalb der Hauptdiagonalen $(i < j)$ zeilenweise gelistet. Ist die Dimension des Problems nicht sehr groß, wird für das Zwischenergebnis ein Vektor bereitgestellt, ansonsten ein File mit Komponenten vom Typ *integer*. Daraus wird das NNE-Textfile abgeleitet. Es hat die Struktur

$isym$	Symmetrie der Matrix (=1, falls Matrix symmetrisch, sonst 0)
n	Dimension der Matrix
nne	Anzahl aller NNE, $nne = n + 2\,nneo$
$nneo$	Anzahl der NNE oberhalb der Hauptdiagonalen
$F(i)$	$i = 1(1)\,2\,nne$, falls $isym = 0$ (Wert $nneo$ ohne Bedeutung)
	enthält die Indizes aller NNE der (nichtsymmetrischen) Matrix
	zeilenweise in geordneter Reihenfolge
	$i_1\ j_1 \quad i_2\ j_2 \quad i_3\ j_3$ ... (je 5 Paare pro Textzeile)
bzw.	
$F(i)$	$i = 1(1)\,2\,(n1 + nneo)$, falls $isym = 1$
	enthält die Indizes NNE der symmetrischen Matrix
	in (Anzahl=$n1 = nne - 2\,nneo \le n$) und **oberhalb** (Anzahl=$nneo$)
	der Hauptdiagonalen zeilenweise in geordneter Reihenfolge
	$i_1 \le j_1 \quad i_2 \le j_2 \quad i_3 \le j_3$ ... (je 5 Paare pro Textzeile)

Die Parameter $isym$ und $nneo$ wurden deshalb mit einbezogen, da natürlich auch solche Matrizen wie a_2 oder symmetrische Matrizen mit Nullelementen auf der Hauptdiagonalen zugelassen sein sollen.
Aus dem NNE-Textfile werden dann unter Beachtung der Symmetrie die Matrixelemente-, Column- und Row-Files A.VAL, A.COL, A.ROW zu A, J bzw. I abgeleitet. Dies sind zwar untypisierte Dateien, aber von wohldefinierter Blockgröße (*double*, *integer*, *longint*). Somit kann man auf ihre Komponenten geeignet zugreifen.

Des weiteren sind die Problemgrößen einfach zu erhalten mittels

$n = filesize(A.ROW) - 1$ (Dateiname=Matrixname=A),

$nne = filesize(A.VAL) = filesize(A.COL)$.

Die Einbeziehung der Dateiarbeit ermöglicht im Programm auch sogenannte "Seiteneinstiege". Liegen die entsprechenden Textdateien für die Vernetzung oder der NNE der Matrix in der geforderten Struktur anderweilig vor, können diese ohne weiteres genutzt werden.

Der oben erwähnte Zwischenschritt erfolgt auf der Basis eines FORTRAN-Programms aus [4]. Dieses wurde auf die Sprache Pascal zugeschnitten und durch die Filearbeit ergänzt.

```
C ---------------------------------------------------------------
C     HP : TRANSFER.FOR    DOS-Version
C     UP : -
C ---------------------------------------------------------------
C     HAUPTPROGRAMM ZUR UEBERFUEHRUNG DER ELEMENTSTRUKTUR EINES FEM-
C     NETZES IN DIE MATRIXBELEGUNG IN FORM VON NICHTNULLELEMENTEN
C     (NNE) ZWECKS WEITERER MINIMIERUNG DER BANDBREITE ODER DES
C     PROFILS
C     DAS PROGRAMM IST AUSGELEGT FUER MAXIMAL
C         NMAX  = 10000 KNOTENPUNKTE
C
C         N     = ANZAHL DER KNOTENPUNKTE, ANZAHL DER ZEILEN DER
C                 MATRIX
C         NMAX  = BEGRENZUNG FUER N
C         M     = ANZAHL DER VON NULL VERSCHIEDENEN MATRIXELEMENTE
C                 OBERHALB DER HAUPTDIAGONALEN
C         MMAX  = BEGRENZUNG FUER M
C
C     EINGABE
C         DATEI  MIT  N, (NKNOT,NP(I),-1/) 0/
C         N
C         NKNOT = ANZAHL DER KNOTENPUNKTE PRO ELEMENT (<=8)
C                 NKNOT = 0 : SCHLUSSZEILE
C         NP(I) = KNOTENNUMMERN PRO ELEMENT
C                 NP(1)<0 : SCHLUSSZEILE FUER ELEMENTTYP
C     AUSGABE
C         DATEI  MIT N, M, A(I)
C         N
C         M
C         A(I), I=1(1)2*M
C                 ENTHAELT DIE INDIZES DER NNE DER MATRIX OBER-
C                 HALB DER HAUPTDIAGONALEN IN GEORDNETER REIHEN-
C                 FOLGE  I1,J1, I2,J2, ...
C ---------------------------------------------------------------
      PARAMETER(NMAX=10000,MMAX=2048,KNOTMAX=8)
      INTEGER  NP(8),A(4098)
      INTEGER  N,M,M2,NKNOT,IN,IZ,I,J,K,L,NNP,NZP
      CHARACTER*48  FNAME1,FNAME3
      WRITE(*,899)
  899 FORMAT(' DATENTRANSFER  ELEMENTE--MATRIX'/)
      WRITE(*,900)
  900 FORMAT('  NAME DER EINGABEDATEI (...ELEM*.DAT) : ')
      READ(*,'(A48)')  FNAME1
      OPEN(UNIT=1,FILE=FNAME1,STATUS='OLD')
      WRITE(*,901)
  901 FORMAT('  NAME DER AUSGABEDATEI (...MATR*.DAT) : ')
```

```
      READ(*,'(A48)')  FNAME3
      OPEN(UNIT=3,FILE=FNAME3,STATUS='UNKNOWN')
      READ(1,*)  N
      IF(N.LT.2  .OR. N.GT.NMAX)  STOP  'N ZU KLEIN/GROSS !!'
C -----------------------------------------------------------------
C     AUFBAU DES NNE-VEKTORS AUFGRUND DER KNOTENNUMMERN DER ELEMENTE
C -----------------------------------------------------------------
      M = 1
      A(1) = N
      A(2) = N+1
   30 READ(1,*)  NKNOT
      IF(NKNOT.GT.KNOTMAX)  STOP  'NKNOT ZU GROSS !!'
      IF(NKNOT.GT.0)  THEN
   40   READ(1,*)  (NP(I), I=1,NKNOT)
        IF(NP(1).LT.0)  GOTO 30
        DO 90 I = 1,NKNOT-1
          NZP = NP(I)
          DO 80 J = I+1,NKNOT
            NNP = NP(J)
            IN = NNP
            IZ = NZP
            IF(NNP.GT.NZP)  GOTO 70
            IN = NZP
            IZ = NNP
   70       K = 1
   72       IF(IZ.GT.A(K))  GOTO 71
            K = K+1
            IF(IZ.LT.A(K-1))  GOTO 52
   62       IF(IZ.EQ.A(K-1) .AND. (IN.GT.A(K)))  GOTO 61
            IF((A(K-1).EQ.IZ) .AND. (A(K).EQ.IN))  GOTO 80
   52       M = M+1
            IF(M.GT.MMAX)  STOP  'M ZU GROSS !!'
            L = 2*M
   42       A(L) = A(L-2)
            A(L-1) = A(L-3)
            L = L-2
            IF(L.GT.K)  GOTO 42
            A(K) = IN
            A(K-1) = IZ
            GOTO 80
   61       K = K+2
            GOTO 62
   71       K = K+2
            GOTO 72
   80     CONTINUE
   90   CONTINUE
        GOTO  40
      ENDIF
C -----------------------------------------------------------------
C     MATRIX IN DATEI
C -----------------------------------------------------------------
      M = M-1
      M2 = 2*M
      WRITE(3,5)  N,M
    5 FORMAT(/1X,2I5)
      WRITE(3,6)  (A(I), I=1,M2)
    6 FORMAT((1X,10I5))
      CLOSE(1)
      CLOSE(3)
      STOP 'S C H L U S S'
      END
```

Beispiel 1: Dreiecksvernetzung eines dreieckigen Gebietes mit Loch.
$isym = 1,\ n = 6,\ nne = 30,\ nneo = 12$

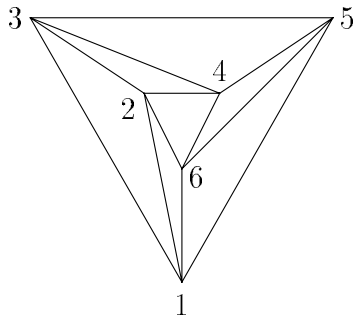

Abb.1. Dreiecksvernetzung

Netz-File **ELEM5.DAT**

```
6
3
1  2  3
1  2  6
2  3  4
3  4  5
5  4  6
5  6  1
-1
0
```

Besetzungsstruktur der
symmetrischen Matrix $a(6,6)$

x	x	x		x	x
	x	x	x		x
		x	x	x	
			x	x	x
				x	x
					x

NNE-File **NMATR5.DAT**

```
1
6   30   12
1  1    1  2    1  3    1  5    1  6
2  2    2  3    2  4    2  6    3  3
3  4    3  5    4  4    4  5    4  6
5  5    5  6    6  6
```

Zwischenvektor der Indexpaare (i,j) der NNE oberhalb der Hauptdiagonalen :
1 2 1 3 1 5 1 6 2 3 2 4 2 6 3 4 3 5 4 5 4 6 5 6

File **A.VAL** der Matrixelemente (NNE $a_{ij} = 1$, GKZ vom Typ *double*) entsprechend dem Vektor $A(1..nne)$:

a_{11} a_{12} a_{13} a_{15} a_{16} a_{21} a_{22} a_{23} a_{24} a_{26}
a_{31} a_{32} a_{33} a_{34} a_{35} a_{42} a_{43} a_{44} a_{45} a_{46}
a_{51} a_{53} a_{54} a_{55} a_{56} a_{61} a_{62} a_{64} a_{65} a_{66}

File **A.COL** der Spaltenindizes der NNE (ganze Zahlen vom Typ *integer*) entsprechend dem Vektor $J(1..nne)$:

1 2 3 5 6 1 2 3 4 6 1 2 3 4 5 2 3 4 5 6 1 3 4 5 6 1 2 4 5 6

File **A.ROW** der Zeilenzeiger der NNE (ganze Zahlen vom Typ *longint*) entsprechend dem Vektor $I(1..n+1)$:

1 6 11 16 21 26 31

Beispiel 2: Gebietsvernetzung mit unterschiedlichen Elementen.
$isym = 1,\ n = 15,\ nne = 117,\ nneo = 51$

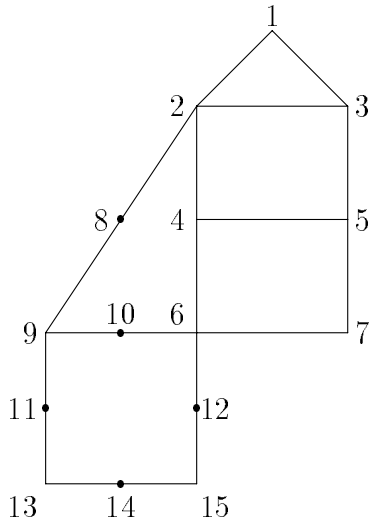

Abb.2. Gebietsvernetzung

Netz-File **ELEM1.DAT**

```
15
3
1  2  3
-1
6
2  9  6  8  10  4
-1
4
2  4  5  3
4  6  7  5
-1
8
9  13  15  6  11  14  12  10
-1
0
```

Besetzungsstruktur der
symmetrischen Matrix $a(15,15)$

```
x x x
  x x x x x   x x x
    x x x
      x x x x x x x
        x x x
          x x x x x x x x x x x
            x
              x x x
                x x x x x x x
                  x x x x x x
                    x x x x x
                      x x x x
                        x x x
                          x x
                            x
```

NNE-File **NMATR1.DAT**

```
1
15   117   51
  1    1    1    2    1    3    2    2    2    3
  2    4    2    5    2    6    2    8    2    9
  2   10    3    3    3    4    3    5    4    4
  4    5    4    6    4    7    4    8    4    9
  4   10    5    5    5    6    5    7    6    6
  6    7    6    8    6    9    6   10    6   11
  6   12    6   13    6   14    6   15    7    7
  8    8    8    9    8   10    9    9    9   10
  9   11    9   12    9   13    9   14    9   15
 10   10   10   11   10   12   10   13   10   14
 10   15   11   11   11   12   11   13   11   14
 11   15   12   12   12   13   12   14   12   15
 13   13   13   14   13   15   14   14   14   15
 15   15
```

Die Struktur des Zwischenvektors und der Files **A.VAL, A.COL, A.ROW** ist analog zum Beispiel 1 und kann aus der Matrixbelegung bzw. dem NNE-File NMATR1.DAT abgelesen werden.

Wir beschränken uns hier auf die Notation des Files **A.ROW** der Zeilenzeiger der NNE (ganze Zahlen vom Typ *longint*) entsprechend dem Vektor $I(1..n+1)$:

1 4 13 18 27 33 46 50 56 67 78 86 94 102 110 118

4.3 Programmiertechnik

Die Kompaktspeicherung der Matrix wird nun integriert in die Matrix-Vektor-Multiplikation $x = a * b$. Für Probleme mittlerer Dimension kann man sich für gewisse Operationen noch im Rahmen des 64KByte-Datensegments bewegen. Wird die Dimension viel größer, muß zusätzlich Filearbeit implementiert werden. Dazu sind abgeleitete Dateien aus der Matrix a sowie weitere Dateien für die Vektoren b und x notwendig.
Das Datensegment sollte man maximal nutzen, um somit den Aufwand für den Datentransfer zwischen externen und Heapspeicher zu minimieren.
In der Unit **DEKLARE.PAS** zum Hauptprogramm **MATVEK1.PAS** (siehe [20]) sind die dafür implementierten Datentypen zu erkennen.

```
{$M 65520}    {Protected-Mode von BP}
{$N+}
unit  Deklare;
{(C)  Neundorf,W. FMN TUI 1996
      Deklarationen
      DEKLARE.PAS

      zum HP  MATVEK1.PAS
              Kompaktspeicherung einer Matrix
              Anwendung bei Matrix-Vektor-Multiplikation }

interface

uses  dos,crt;

const nmax = 8191;    { 8191 = maximale Dimension der Matrix a(n,n)
                        64KByte-Datensegment = 65536Byte
                                             = 8*8192 double-Komponenten }
      nknotmax = 50;  { max. Anzahl von Knoten pro Element }
      kmax = 170;     { Grenze in FEM bei Vektor/File fuer Netz --> NNE }

type  float   =double; { real,single,double,extended }
      vektor  =array[1..nmax] of float;
      column  =array[1..nmax] of integer;
      row     =array[1..nmax+1] of longint;
      knvektor=array[1..nknotmax] of integer;
      pvektor =^vektor;
      pcolumn =^column;
      prow    =^row;

const size_el    =sizeof(float);
      size_row_el=sizeof(longint);
      size_col_el=sizeof(integer);

var   a_values    :pvektor;    { von Matrix a abgeleitete }
      a_column    :pcolumn;    { Hilfsvektoren im Heap  }
      a_row       :prow;       { fuer A,J,I }
      b_values,x  :pvektor;
      matrixfile  :file; {of float}     { *.VAL }
      columnfile  :file; {of integer}   { *.COL }
      rowfile     :file; {of longint}   { *.ROW }
      b_vektorfile:file; {of float}
      x_vektorfile:file; {of float}
      n           :integer;    { Dimension der Matrix a, Knotenpunktanzahl }
      nne,                     { Anzahl aller NNE }
      nneo        :longint;    { Anzahl der NNE oberhalb der Hauptdiag. }
```

```
      version,                 { Versionen zur Behandlung der von a
                                 abgeleiteten Files fuer  A,J,I }
      isym        :byte;       { Symmetrie von a (isym=1 --> a symm.) }

      matrixname,b_vektorname,x_vektorname,
      netzfname,nnefname:string[8];
      netzfile,nnefile  :text; { Textfiles : *ELEM*.DAT  Netz-File
                                             *MATR*.DAT  NNE-File  }
      eina,einb,berx:boolean;  { Steuergroessen }

implementation

end.
```

Neben dieser Unit mit den wichtigsten Vereinbarungen enthält das Hauptprogramm noch die Units :

- **MATVEKU1.PAS** UP zur Dateneingabe bei kleinen Systemen, $n \leq 10$,
- **MATVEKU2.PAS** UP zur Dateneingabe bei FEM-Systemen,
  Ein-/Vorgabe der Gebietsvernetzung,
  Ein-/Vorgabe der NNE-Matrix.

Die Komponentengröße $float = double$, $integer$, $longint$ der untypisierten Files $matrixfile$, $columnfile$, $rowfile$ für die Vektoren A, J bzw. I wird mittels $sizeof()$ erfragt.
File-Speicher-Datentransfer erfolgt mittels $blockwrite$, $blockread$.

4.3.1 Versionen der Dateiarbeit

In der Unit DEKLARE.PAS ist der Auswahlparameter ***version*** zu erkennen. Er dient zur Festlegung der Version der Filebehandlung und korrespondiert mit dem Verhältnis von n zu $nmax$.

Version Nr.	Verhältnis der Dimensionen	Bemerkung
1	$n \leq nmax$	mit Vektortypen der Länge $1..nmax(+1)$ sind die Inhalte der Files für I, b, x mit **einem** Transfer komplett auf Vektoren im Heap übertragbar, Files für A, J i.a. nur abschnittsweise lesbar
2	$n \geq nmax$	Files für A, J, I blockweise transferiert, b, x komponentenweise verarbeitet
3	$n \geq nmax$	Files für A, J, I, b, x blockweise transferiert, Blockgröße=$nmax$

Tab.8. Beschreibung der Versionen der Dateiarbeit

(1) Eröffnung der typenlosen Dateien mit entsprechender Blockgröße/Datenformat und Abfrage der Problemdimensionen

```
assign(matrixfile,matrixname+'.VAL');
reset (matrixfile,size_el);
assign(columnfile,matrixname+'.COL');
reset (columnfile,size_col_el);
nne:=filesize(matrixfile);
assign(rowfile,matrixname+'.ROW');
reset (rowfile,size_row_el);
n:=filesize(rowfile)-1;
assign(b_vektorfile,b_vektorname+'.VAL');
reset (b_vektorfile,size_el);
assign (x_vektorfile,x_vektorname+'.VAL');
rewrite(x_vektorfile,size_el);
```

(2) Version 1

Die Handbabung der Elemente von I, b, x als Komponenten von Vektoren, die vollständig im Heap abgelegt sind, ist einfach zu durchschauen. Der Zugriff auf die Komponenten der Files für A, J erfolgt im "Gleichschritt", wobei immer nur Blöcke von $nmax$ Komponenten in die entsprechenden Heapvektoren geladen werden (letzter Block eventuell kürzer).

Der fortlaufende Index der Ergebniskomponente ergibt sich auf der Basis des Zeilenzeigervektors. Dabei ist zu berücksichtigen, daß in manchen Zeilen der Matrix nur Nullelemente stehen können. Der äußere Zyklus wird gesteuert durch die Anzahl der NNE.

```
1 : begin
     for i:=1 to n do x^[i]:=0;
     blockread(matrixfile,a_values^,nmax,result);
     blockread(columnfile,a_column^,nmax,result);
     blockread(rowfile,a_row^,n+1,result1);
     blockread(b_vektorfile,b_values^,n,result2);
     val_count:=0;
     k:=0;
     i:=1;
     repeat
      inc(val_count);
      inc(k);
      while k=a_row^[i+1] do inc(i);
      j:=a_column^[val_count];

      x^[i]:=x^[i]+a_values^[val_count]*b_values^[j];

      if val_count=result then
       begin
        blockread(matrixfile,a_values^,nmax,result);
        blockread(columnfile,a_column^,nmax,result);
        val_count:=0;
       end;
     until k=nne;
     blockwrite(x_vektorfile,x^,n);
    end;
```

(3) Version 2

Für A, J, I bleiben wir bei der Verarbeitung mit Blöcken der Länge $nmax(+1)$. Der äußere Zyklus wird gesteuert durch den Zeilenzeigervektor und wird in ungefähr $\frac{n}{nmax}$ Zyklen zerlegt. Der Grund dafür ist, daß I den Zeilenwechsel bestimmt und mit jedem Zeilenwechsel der Vektor b zurückzusetzen ist (`reset(b_vektorfile,size_el)`).

$nmax$	1	2	3	4
Zyklen	1 3 3 5 5 7 7 9 9 12	1 3 5 5 7 9 9 12	1 3 5 7 7 9 12	1 3 5 7 9 9 12
Anzahl $izykl$	5	3	2	2

Tab.9. Zyklen und ihre Anzahl auf der Basis des Vektors I,
Matrix $a_2(5,5)$ aus Kap. 4.1, $I(1..6) = (1,3,5,7,9,12)$, $nne = 11$

Die Zyklenanzahl $izykl$ wird berechnet durch Aufrunden des Wertes

$$h = \frac{(n+1)(nmax+1)-2}{nmax(nmax+1)}$$ auf die nächste ganze Zahl.

In jedem Zyklus gibt es maximal $nmax$ innere Schritte. Zu Beginn eines solchen ist die x-Komponente zu initialisieren und der Vektor b zurückzusetzen. Gibt es nur Nullelemente in der Matrixzeile (`diff=0`), dann wird die x-Komponente zu Null gesetzt. Ansonsten muß diese durch Aufdatieren von $a_{ij} * b_j$ (`a_values^[k]*bb`) aktualisiert werden. Dabei kann es passieren, daß weitere Blöcke von A, J geholt werden müssen (`blockread(matrixfile,...)`; `blockread(columnfile,...)`).

```
2 : begin
     blockread(matrixfile,a_values^,nmax,result);
     blockread(columnfile,a_column^,nmax,result);
     h:=((n+1.0)*(nmax+1)-2)/(nmax*(nmax+1));
     if frac(h)<1E-6 then izykl:=round(h)
                     else izykl:=trunc(h)+1;
     blockread(rowfile,aa,1,result1);
     k:=0;

     for i:=1 to izykl do
      begin
       blockread(rowfile,a_row^,nmax,result1);
       for l:=result1 downto 1 do a_row^[l+1]:=a_row^[l];
       a_row^[1]:=aa;
       aa:=a_row^[result1+1];
       for val_count:=1 to result1 do
        begin
```

```
          xx:=0;
          reset(b_vektorfile,size_el);
          diff:=a_row^[val_count+1]-a_row^[val_count];
          if diff=0 then blockwrite(x_vektorfile,xx,1)
                    else
           begin
            je:=0;
            for m:=1 to diff do
             begin
              inc(k);
              j:=a_column^[k];
              for j1:=1 to j-je do blockread(b_vektorfile,bb,1);
              je:=j;

              xx:=xx+a_values^[k]*bb;

              if k=result then
               begin
                blockread(matrixfile,a_values^,nmax,result);
                blockread(columnfile,a_column^,nmax,result);
                k:=0;
               end;
             end;
            blockwrite(x_vektorfile,xx,1);
           end; { diff<>0 }
         end;
       end;
     end;
```

(4) Version 3

Im Vergleich zur Version 2 können wir Filetransfer dadurch einsparen, wenn
- b **nicht** komponentenweise geladen,
- x **nicht** komponentenweise gespeichert

werden.

Wenn die entsprechenden Heapvektoren von b und x abgearbeitet bzw. aufgefüllt worden sind, ist ersterer neu zu belegen bzw. das Ergebnis abzuspeichern. Dieser Zusatz sowie die eventuell besondere Situation im allerletzten Zyklus machen im wesentlichen den Mehraufwand aus.

```
3 : begin
     blockread(matrixfile,a_values^,nmax,result);
     blockread(columnfile,a_column^,nmax,result);
     h:=((n+1.0)*(nmax+1)-2)/(nmax*(nmax+1));
     if frac(h)<1E-6 then izykl:=round(h)
                     else izykl:=trunc(h)+1;
     h1:=(n+izykl) mod (nmax+1) -1;
     blockread(rowfile,aa,1,result1);
     k:=0;
     ii:=0;

     for i:=1 to izykl do
      begin
       blockread(rowfile,a_row^,nmax,result1);
       for l:=result1 downto 1 do a_row^[l+1]:=a_row^[l];
```

```
   a_row^[1]:=aa;
   aa:=a_row^[result1+1];
   for val_count:=1 to result1 do
    begin
     xx:=0;
     reset(b_vektorfile,size_el);
     blockread(b_vektorfile,b_values^,nmax,result2);
     diff:=a_row^[val_count+1]-a_row^[val_count];
     if diff=0 then
      begin
       inc(ii);
       x^[ii]:=xx;
       if (ii=nmax) or
          (ii=h1) and (i=izykl) and (val_count=result1) then
        begin
         blockwrite(x_vektorfile,x^,ii);
         ii:=0;
        end;
      end
               else
      begin
       je:=0;
       for m:=1 to diff do
        begin
         inc(k);
         j:=a_column^[k];
         if j<=je+result2 then bb:=b_values^[j-je]
                          else
          begin
           repeat
            inc(je,result2);
            blockread(b_vektorfile,b_values^,nmax,result2);
           until j<=je+result2;
           bb:=b_values^[j-je];
          end;

         xx:=xx+a_values^[k]*bb;

         if k=result then
          begin
           blockread(matrixfile,a_values^,nmax,result);
           blockread(columnfile,a_column^,nmax,result);
           k:=0;
          end;
        end;
       inc(ii);
       x^[ii]:=xx;
       if (ii=nmax) or
          (ii=h1) and (i=izykl) and (val_count=result1) then
        begin
         blockwrite(x_vektorfile,x^,ii);
         ii:=0;
        end;
      end; { diff<>0}
    end; { for val_count }
  end; { for i }
end;
```

4.4 Testbeispiele

(1) TB1
Multiplikation einer voll besetzten Matrix $a(n,n)$ mit Vektor $b(n)$, $b_i = 1$.
Die Zeitkomplexität des Algorithmus beträgt $T(n) = \mathcal{O}(n^2)$.

(2) TB2

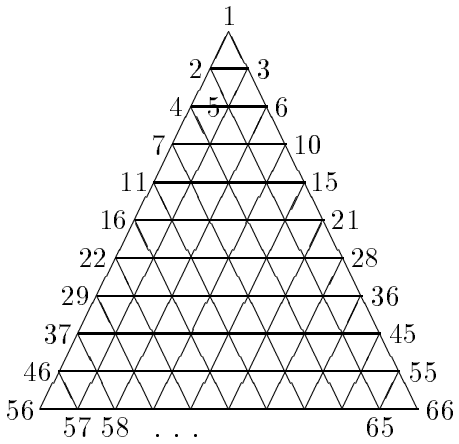

Abb.3. Dreieckvernetzung
Dreieckelemente mit linearem Ansatz
Knotenpunkte zeilenweise durchnumeriert
$isym = 1$
$n = 66,\ nne = 396,\ nneo = 165$

(3) TB3 (dieses und weitere Beispiele aus [4])

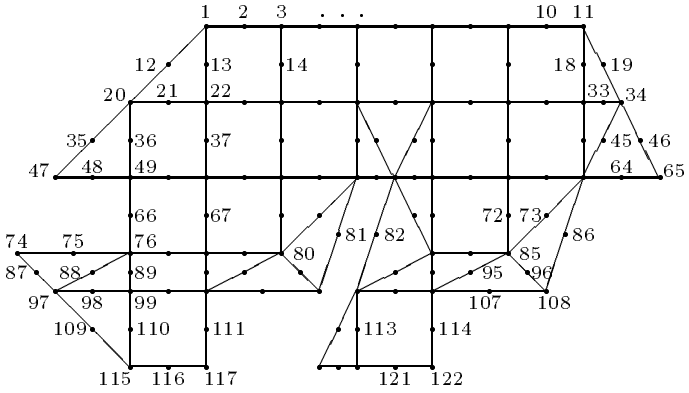

Abb.4. Längsschnitt
Autoinnenraum
Dreieck- und Rechteckelemente
mit quadratischem Ansatz
Knotenpunkte zeilenweise
durchnumeriert
$isym = 1$
$n = 122,\ nne = 1390,\ nneo = 634$

(4) TB4

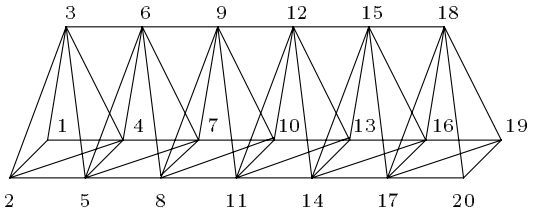

Abb.5. Statisches Fachwerk
Kranträger
Stabelemente
Knotenpunkte abschnittsweise
durchnumeriert
$isym = 1$
$n = 20,\ nne = 128,\ nneo = 54$

(5) TB5,6,7

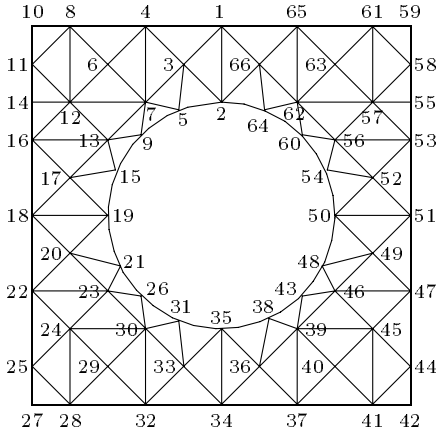

Abb.6. Elliptische RWA
Ebenes Gebiet
mit Öffnung/Loch
Dreieck- und Rechteckelemente
mit linearem/bilinearem Ansatz
Knotenpunkte in positiver
Umlaufrichtung durchnumeriert
$isym = 1$
$n = 66,\ nne = 378,\ nneo = 156$

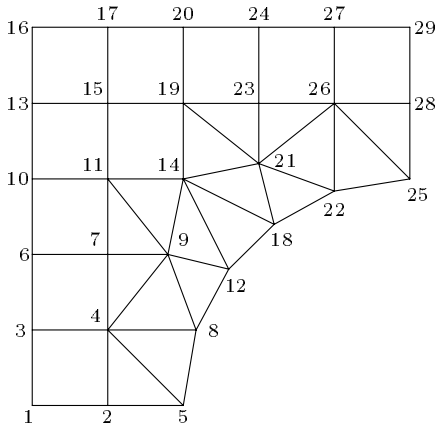

Abb.7. Elliptische RWA
Gebiet mit Öffnung/Loch
wegen Symmetrie Viertelgebiet
Dreieck- und Rechteckelemente
mit linearem/bilinearem Ansatz
Numerierung der Knotenpunkte
zwecks günstiger Bandstruktur
$isym = 1$
$n = 29,\ nne = 179,\ nneo = 75$

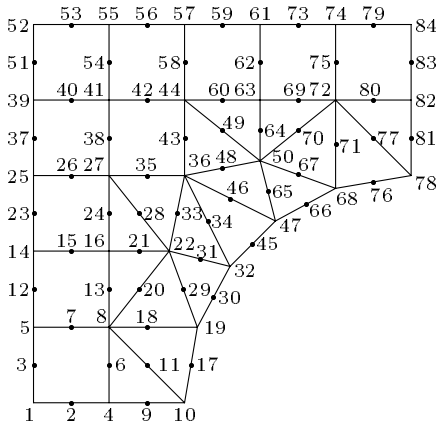

Abb.8. Elliptische RWA
Gebiet mit Öffnung/Loch
wegen Symmetrie Viertelgebiet
geradlinige Elemente
mit quadratischem Ansatz
Numerierung der Knotenpunkte
zwecks günstiger Bandstruktur
$isym = 1$
$n = 84,\ nne = 938,\ nneo = 427$

(6) TB8,9

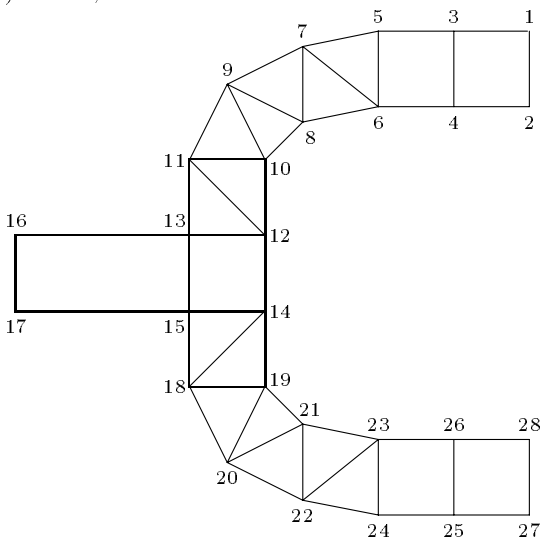

Abb.9. Stimmggabel
Eigenfrequenzen und
Eigenschwingungsformen
Dreieck- und Rechteckelemente
mit linearem/bilinearem Ansatz
Numerierung der Knotenpunkte
zwecks günstiger Bandstruktur
$isym = 1$
$n = 28,\ nne = 146,\ nneo = 59$

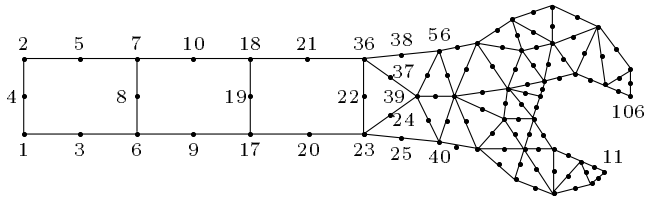

Abb.10. Gabelschlüssel
Deformation einer Scheibe
Dreieck- und Rechteckelemente
mit quadratischem Ansatz
Numerierung der Knotenpunkte
zwecks günstiger Bandstruktur
$isym = 1$
$n = 106,\ nne = 1036,\ nneo = 465$

Rechenzeitvergleiche am PC

Die Rechenzeitangaben erfolgen zwar mit Hunderstelsekunden, aber durch Zeitschwankungen von ungefähr 2% ist die Relevanz der Angaben von Nachkommastellen in manchen Fällen entsprechend gemindert. Die Matrixelementesind vom Format *double*.

n	Anzahl der Blocktransfer für A, J bei $nmax =$					Rechenzeit $t(nmax)$ $[sec]$ für $nmax =$				
	8191	8000	6000	4000	2000	8191	8000	6000	4000	2000
100	2	2	2	3	5	0.00	0.00	0.00	0.00	0.05
500	31	32	42	63	125	1.64	2.20	2.47	2.47	2.52
1000	123	125	167	250	500	6.15	8.78	9.17	9.72	9.72
1500	275	282	375	563	1125	13.62	19.33	22.90	22.94	22.96
2000	489	500	667	1000	2000	24.44	34.72	36.58	39.10	39.76
4000	1954	2000	2667	4000		117.10	146.54	150.44	154.51	

Tab.10. Testbeispiel TB1, voll besetzte Matrix $a(n,n)$, $nne = n^2 > nmax \geq n$,
Blocktransfer und Rechenzeiten der Version 1 auf PC i-Pentium in sec

Zwischen den Varianten $nmax = 6000, 4000, 2000$ sind keine signifikanten Unterschiede in der Rechenzeit, d.h. der Filetransferaufwand ist fast der gleiche bei diesen Blockgrößen. Erstaunlich ist der Rechenzeitgewinn, wenn die maximale Blockgröße $nmax = 8191$ eingestellt ist. Ein Grund für das Abknicken der Zeitfunktion mit Erreichen dieser Grenze wird in einer besonderen inneren Speicherkonstellation gesehen.

$nmax$	n	Anzahl der Blocktransfer für A, J	Rechenzeiten der Versionen		
			1	2	3
4000	4000	4000	154.51	$\approx$ 30 *min*	154.56
2000	100	5	0.05		
	500	125	2.52		
	1000	500	9.72		
	1500	1125	22.96		
	2000	2000	39.76	577.49	45.64
	4000	8000		$\approx$ 36 *min*	179.71
1500	100	7	0.05		
	500	167	2.69		
	1000	667	10.44		
	1500	1500	23.75	325.76	25.38
	2000	2667		576.99	47.17
	4000	10667		$\approx$ 36 *min*	182.51
1000	100	10	0.05		
	500	250	2.69		
	1000	1000	10.55	146.05	11.90
	1500	2250		321.10	26.15
	2000	4000		578.40	47.28
	4000	16000		$\approx$ 38 *min*	193.67
500	100	20	0.05		
	500	500	2.75	36.69	3.29
	1000	2000		146.26	11.98
	1500	4500		328.84	27.08
	2000	8000		$\approx$ 10 *min*	49.40
	4000	32000		$\approx$ 46 *min*	229.59
100	100	100	0.06	1.48	0.16
	500	2500		36.97	4.28
	1000	10000		147.36	16.21
	1500	22500		$\approx$ 6 *min*	36.30
	2000	40000		$\approx$ 10 *min*	64.32
	4000	160000			301.41

Tab.11. Testbeispiel TB1, voll besetzte Matrix $a(n, n)$, $nne = n^2 > nmax$
Rechenzeiten der Versionen 1,2,3 auf PC i-Pentium in *sec*

Die Durchführung zulässiger Testrechnungen zeigt, daß sich insbesondere die Begrenzung des Filetransfers günstig auf die Rechenzeit auswirkt. Sobald noch eine Partitionierung der Datenfiles für I, b, x wegen zu kleinem Datenvektors im Heap oder eine komponentenweise Verarbeitung von b, x notwendig sind, wirkt sich dies mit erheblichen Zeitverlusten aus. Letztere, also Version 2, ist ungefähr 12 Mal langsamer als Version 3 (bei $nmax = 100$ ca. 9 Mal).

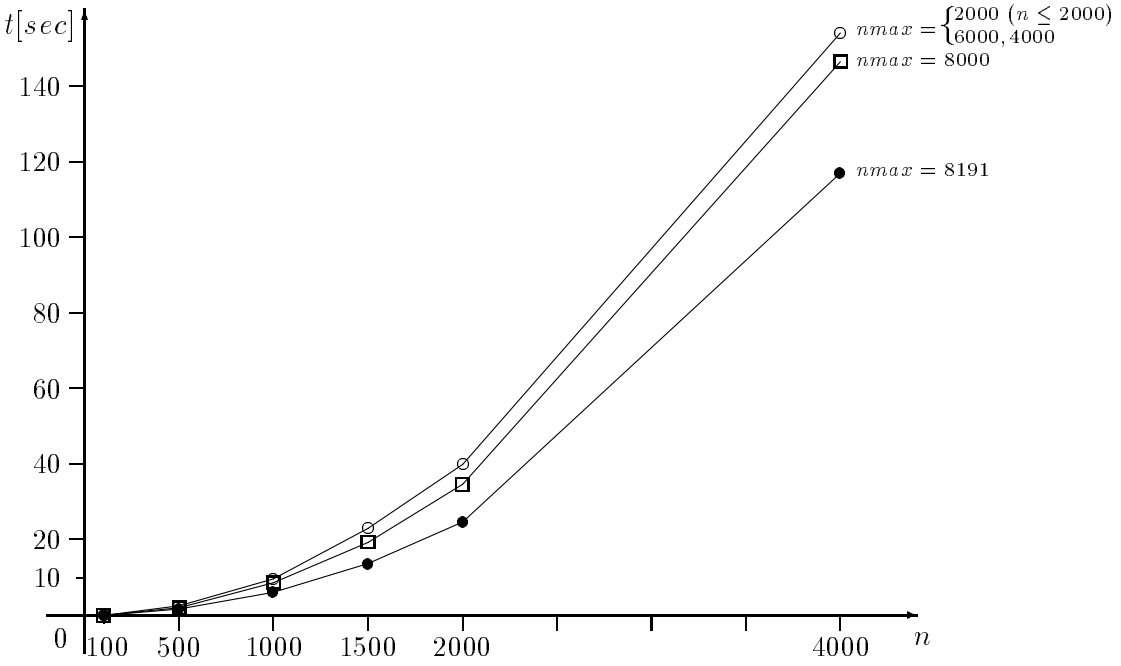

Abb.11. Rechenzeiten $t(n)$ in sec der Version 1 auf PC i-Pentium für $nmax = 8191, 8000(-2000)2000$

Aufgrund der mit wachsendem n auch quadratisch steigenden Anzahl der Blocktransfer verändert sich die Zeitkomplexitätsfunktion zu $T(n, nmax) = \mathcal{O}(n^2 + nmax)$. Analoges Verhalten finden wir auch bei der Version 3.

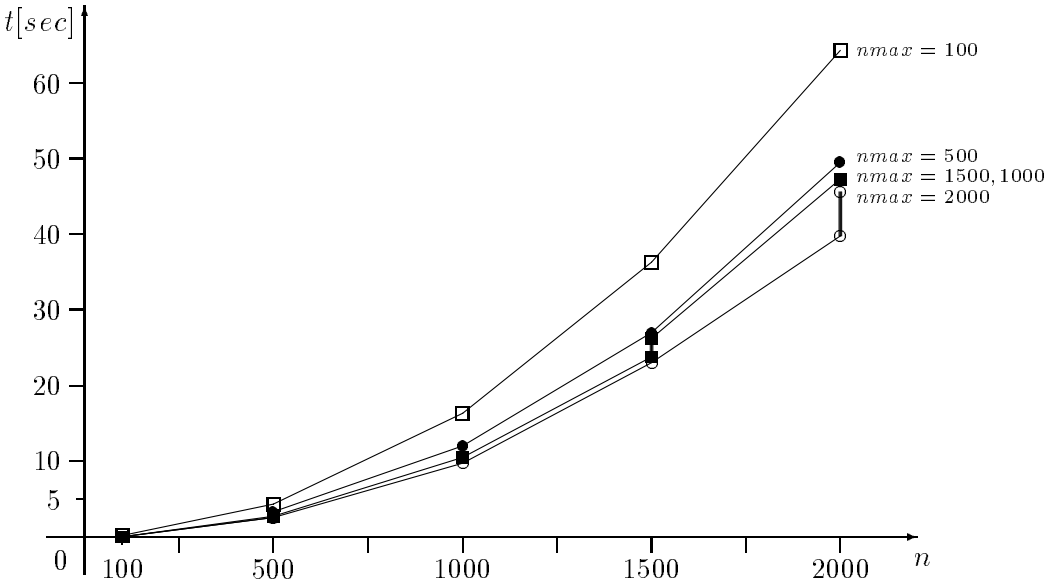

Abb.12. Rechenzeiten $t(n)$ in sec der Versionen 1 ($n \leq nmax$) und 3 ($n \geq nmax$) auf PC i-Pentium für $nmax = 2000(-500)500,\ 100$

An den Stellen $n = nmax$ treten beim Übergang V1$\rightarrow$V3 Sprungstellen auf, die in der Graphik zumindest für $nmax = 2000, 1500$ als vertikale Strecken zu erkennen sind.

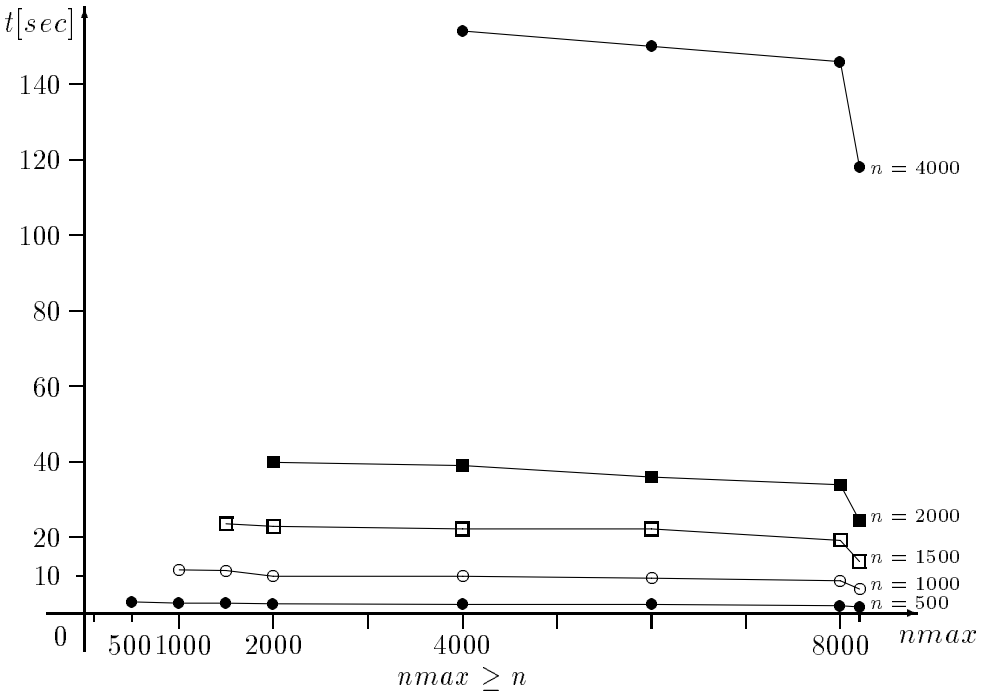

Abb.13. Rechenzeiten $t(nmax)$ in sec der Version 1 auf PC i-Pentium für $n \leq nmax$
Die Größe $nmax$ des Heap-Vektors hat nur ganz geringe Auswirkung auf die Rechenzeit bei kleinem n. Für $n > 2000$ werden Zeitgewinne mit wachsendem $nmax$ deutlicher. Auf den "Knick" wurde in der Bemerkung zur Tabelle 10 schon verwiesen.

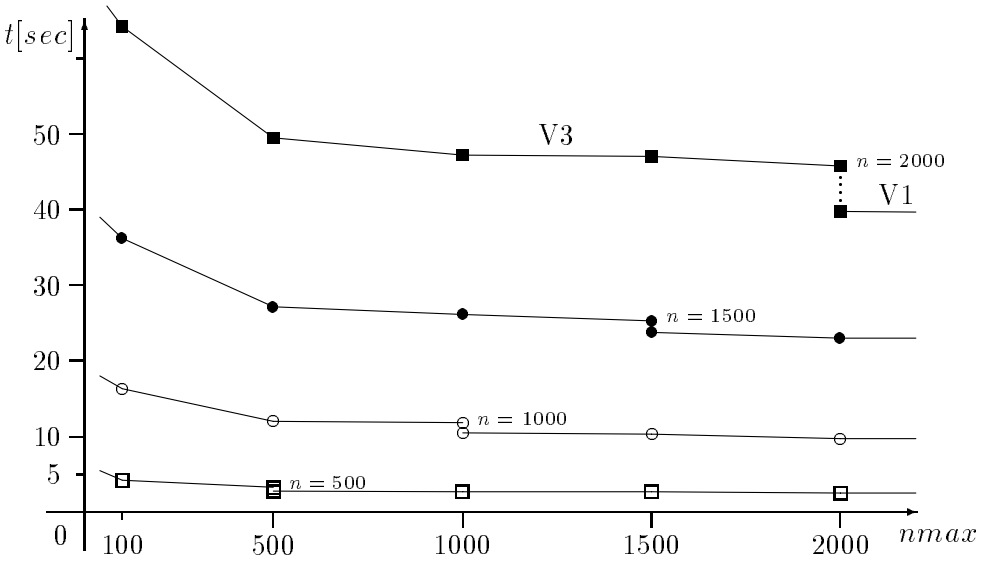

Abb.14. Rechenzeiten $t(nmax)$ in sec der Versionen 3 ($n \geq nmax$) und 1 ($n \leq nmax$) auf PC i-Pentium für verschiedene n

Im ersten Abschnitt der Funktion $t(nmax)$ ist eine stärkere Abnahme zu verzeichnen. Der abfallende Verlauf ist insgesamt gut ausgeprägt bei $n = 4000$ (siehe Tab. 11) wie auch für $nmax < 100$ (hier nur angedeutet). Aber er verliert sich schnell mit kleiner werdendem n.

TBm m	n nne	Version 1 $nmax=$ nne	Version 1 $nmax=$ n	V.2 $nmax$ n	V.3 $nmax$ n	Versionen 2,3 $nmax=$ 1	2	5	10	20	
Kran	20	0.00	0.00	0.11	0.05	0.11	0.11	0.11	0.11	0.11	V2
4	128					0.16	0.11	0.06	0.05	0.05	V3
Stimmg.	28	0.00	0.00	0.16	0.11	0.17	0.16	0.16	0.16	0.16	V2
8	146					0.16	0.16	0.11	0.11	0.11	V3
RWA1	29	0.00	0.00	0.16	0.11	0.22	0.16	0.16	0.16	0.16	V2
6	179					0.22	0.16	0.11	0.11	0.11	V3
Lochgeb.	66	0.00	0.00	0.49	0.22	0.66	0.61	0.61	0.55	0.55	V2
5	378					0.60	0.44	0.33	0.28	0.27	V3
Dreieck	66	0.00	0.00	0.55	0.27	0.66	0.60	0.55	0.55	0.55	V2
2	396					0.66	0.44	0.33	0.33	0.28	V3
RWA2	84	0.03	0.05	0.78	0.38	1.10	0.99	0.88	0.82	0.82	V2
7	938					1.05	0.71	0.49	0.44	0.39	V3
Schlüssel	106	0.05	0.05	1.19	0.49	1.65	1.48	1.32	1.21	1.21	V2
9	1036					1.59	1.04	0.72	0.60	0.49	V3
Auto	122	0.05	0.06	1.59	0.60	2.14	1.92	1.70	1.65	1.61	V2
3	1390					2.09	1.32	0.93	0.71	0.65	V3

TBm m	n nne	Versionen 1,2,3 $nmax=$ 30,40	60	80	100	120	200	
Lochgeb.	66	-	-	0.00	0.00	0.00	0.00	V1
5	378	0.55	0.49	-	-	-	-	V2
		0.27	0.22	-	-	-	-	V3
Dreieck	66	-	-	0.00	0.00	0.00	0.00	V1
2	396	0.55	0.55	-	-	-	-	V2
		0.27	0.27	-	-	-	-	V3
RWA2	84	-	-	-	0.05	0.05	0.04	V1
7	938	0.82	0.77	0.77	-	-	-	V2
		0.39	0.38	0.38	-	-	-	V3
Schlüssel	106	-	-	-	-	0.05	0.05	V1
9	1036	1.21	1.21	1.21	1.20	-	-	V2
		0.49	0.50	0.50	0.49	-	-	V3
Auto	122	-	-	-	-	-	0.05	V1
3	1390	1.60	1.59	1.59	1.59	1.59	-	V2
		0.60	0.60	0.60	0.60	0.60	-	V3

Tab.12. Testbeispiele TB2..9, FEM-Matrix $a(n,n)$,
Rechenzeiten $t(n,nmax)$ der Versionen 1,2,3 auf PC ASI in sec,
Borland Pascal im Protected Mode von DOS-Oberfläche gestartet

Es gilt $t \approx 0$ für $n < 84$ und $n \leq nmax$ (Version 1). Die Zeiten der Versionen 2 und 3 nähern sich an bei $nmax \to 1$. Bei festem n bleiben für eine breiten Bereich von $nmax = n/5...n$ die Zeiten bei V2 bzw. V3 annähernd konstant.

Noch einige Bemerkungen zur Zeitkomplexitätsfunktion $T(n, nmax)$.
Dabei sollen hier der Knick bei der Version 1 nahe $nmax = 8191$ sowie die Version 2 außer Acht gelassen werden. Wir haben die zwei grundsätzlichen Fälle.

- Testbeispiele TB2...9 (FEM-Matrix)
  Die Anzahl nne der NNE ist abgesehen von $nne \leq n^2$ unabhängig von n. Das führt auf die Komplexität $T(nne, nmax) = \mathcal{O}(nne + nmax)$.
  Für festes $nmax$ ist $T(nne, nmax) = c_V\, nne$, wobei sich der positive Parameter c_V für die Versionen 1 und 3 unterscheidet. Weiterhin hat die Struktur der Matrix schon einen gewissen Einfluß, z.B. durch die Umstand, daß viele Nullzeilen in der Matrix auftreten können.
  Für festes nne ist $T(nne, nmax)$ eine monoton fallende Funktion in $nmax$. Dabei gehört sie in der ersten Phase $nmax = 1..n$ zur Version 3 und hat einen Verlauf ähnlich zur Funktion $\alpha + \beta\, e^{-\gamma\, nmax}$. Im zweiten Abschnitt $nmax \geq n$ verhält sie sich wie eine allmählich abnehmende lineare Funktion (siehe auch Abb. 14).

- Testbeispiel TB1 (volle Matrix)
  Hier ist $nne{=}n^2$. Die Anzahl der Blocktransfer wächst bei festem $nmax$ quadratisch mit n. Die Komplexität ist
  $$T(n, nmax) = \mathcal{O}(n^2 + nmax) = c\, n^2 - c_V\, nmax, \quad c, c_V > 0.$$
  Die bisherigen Auswertungen basierten auf folgenden Werten.

	Version 1 $nmax \geq n$							
Version 3	$nmax =$							
$nmax \leq n$	100	500	1000	1500	2000	4000	6000	8000
100	0.06	0.05	0.05	0.05	0.05	0.00	0.00	0.00
500	0.16	2.75	2.69	2.69	2.52	2.47	2.47	2.20
1000	4.28	3.29	10.55	10.44	9.72	9.72	9.17	8.78
1500	16.21	11.98	11.90	23.75	22.96	22.94	22.90	19.33
2000	36.30	27.08	26.15	25.38	39.76	39.10	36.58	34.72
4000	64.32	49.40	47.28	47.17	45.64	154.51	150.44	146.54
n	301.41	229.59	193.67	182.51	179.71	154.56		

Tab.13. Testbeispiel TB1, voll besetzte Matrix $a(n, n)$,
Rechenzeiten $t(n, nmax)$ der Versionen 1,3 auf PC i-Pentium in *sec*

Sie lassen auf eine Funktion der Gestalt
$$T(n, nmax) = \begin{cases} c_1\, n^2 - c_{V1}\, nmax, & bei\ Version\ 1 \\ c_3\, n^2 - c_{V3}\, nmax, & bei\ Version\ 3 \end{cases}$$
schließen. Für festes n ist in V3 ($nmax \leq n$) der Koeffizient c_{V3} eine Funktion von $nmax$ und anschließend in V1 ($nmax \geq n$) der Koeffizient $c_{V1} \ll 1$. $T(n, nmax)$, $n = nmax$, in beiden Versionen liefert die kleinen Sprungstellen in den Abb. 12,14.
In der Version 1 muß $T(n, nmax)$ mit wachsendem $nmax$ gegen eine positive Grenzfunktion $T(n)$ (quadratische Parabel) streben.
Falls $nmax1 > nmax$ ist, dann gilt $T(n, nmax1) < T(n, nmax)$, wobei bei festem $\Delta nmax = nmax1 - nmax$ und wachsendem $nmax$ die Abstände zwischen den Funktionswerten wegen dem Grenzwert immer kleiner werden. Allein der schon erwähnte Knick stört ein wenig dieses Konzept.

5 Schlußbemerkung

Gegenstand der Untersuchungen waren Fragen der ökonomischen Speicherung für große Matrizen und des Aufwands bei Zugriff auf diese bei Speicherung im Zentralspeicher, im Heap oder auf Dateien. Das weitestgehende Ausnutzen des Heaps eröffnet schon zusätzliche Möglichkeiten und bringt Aufwandseinsparungen.
Auf spezielle Techniken zur Ausnutzung einer Band-, Block- oder hüllenartigen Struktur der Matrix ist nicht eingegangen worden. Interessant sind in diesem Zusammenhang auch Strategien der "ungepackten oder gepackten Diagonalen", die ausschließlich Diagonalen der Matrix mit NNE speichern, und dabei die jeweilige Diagonale komplett, falls sie mehr als zu 2/3 mit NNE belegt ist, oder kompakt (nur die wenigen NNE) im anderen Fall (siehe [7]).

Literatur

[1] KIELBASINSKI, A.; SCHWETLICK, H.: *Numerische lineare Algebra.* Mathematik für Naturwissenschaft und Technik Band 18, DVW, Berlin 1988.

[2] HACKBUSCH, W.: *Iterative Lösung großer schwach besetzter Gleichungssysteme.* Leitfäden der angewandten Mathematik und Mechanik Band 69. B.G. Teubner Stuttgart 1991.

[3] MAESS, G.: *Vorlesungen über numerische Mathematik.* Band 1, 2. Akademie-Verlag Berlin 1984, 1988.

[4] SCHWARZ, H.R.: 1.*Methode der finiten Elemente.* Leitfäden der angewandten Mathematik und Mechanik Band 47. B.G. Teubner Stuttgart 1991.
2. *FORTRAN-Programm zur Methode der finiten Elemente.* B.G. Teubner Stuttgart 1991.

[5] ZLATEV, Z.: *Computational Methods for General Sparse Matrices.* Math. and Its Appl. Vol.65. Kluwer Academic Publishers London 1991.

[6] GUSTAVSON, F.: *A Survey of Some Sparse Matrix Theory and Techniques.* Jahrbuch Überblicke Mathematik. B.I.-Wissenschaftsverlag Mannheim 1981.

[7] SCHMAUDER, M.; WEISS, R:; SCHÖNAUER, W.: *The CADSOL Program Package* (Version 1.1). Interner Bericht Nr. 46/92, RZ der Universität Karlsruhe 1992.

[8] SCHWETLICK, H.; KRETZSCHMAR, H.: *Numerische Verfahren für Naturwissenschaftler und Ingenieure.* Fachbuchverlag Leipzig Köln 1991.

[9] BRAMDLER, A.; ALLAN, R.N.; HAMANN, Y.M.: *Sparsity.* Pitman Publishing London 1976.

[10] SCHENDEL, U.: *Sparse Matrizen.* Oldenbourg Verlag München/Wien 1976.

[11] KÖCKLER, N.: *Numerische Algorithmen in Softwaressystemen : unter besonderer Berücksichtigung der NAG-Bibliothek.* B.G. Teubner Stuttgart 1990.

[12] RICE, J.R.: *Numerical Methods, Software and Analysis.* 2nd Edition. Academic Press Inc. Boston 1993.

[13] GOVAERTS, W.; PRYCE, J.D.: *Mixed block elimination for linear systems with wide borders.* IMA Journ. of Numerical Analysis (1993)13,161-180.

[14] COLLINS, R.J.: *Bandwith Reduction by Automatic Renumbering.* Int. Journ. Num. Methods in Engineering 6(1973)345-356.

[15] GIBBS, N.E.; POOLE, W.G.; STOCKMEYER, P.K.: *An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix.* SIAM Journ. Numerical Analysis 13(1976)2,236-250.

[16] BERGER, C.: *Entwurf und Implementierung dünn besetzter Blockmatrizen in C++.* Diplomarbeit TU München IfI 1994.

[17] SPIESS, J.: *Untersuchungen des Zeitgewinns durch neue Algorithmen zur Matrix-Multiplikationen.* Computing 17, 23-36 (1976).

[18] NEUNDORF, W.; ORTLEPP, TH.: *Berechnung von Matrix-Multiplikationen auf dem PC.* Preprint No. M 15/95, TUIlmenau FMN IfMath August 1995.

[19] NEUNDORF, W.: *Pascal-Programm* INV_AUSP.PAS .
Invertierung einer quadratischen Matrix mittels Austauschverfahren mit Spaltenpivotsuche und Zeilenvertauschung (Gauß-Jordan) sowie mit Pointertechnik. TU Ilmenau 1995.

[20] NEUNDORF, W.; HOHLBEIN, D.: *Pascal-Programm* MATVEK1.PAS .
Kompaktspeicherung einer Matrix und Anwendung bei Matrix-Vektor-Multiplikation. TU Ilmenau 1996.

Anschrift:

Dr. Werner Neundorf
Institut für Mathematik
Technische Universität Ilmenau
PF 10 0565
D - 98684 Ilmenau

e-mail : neundorf@mathematik.tu-ilmenau.de